Encoding a dependent-type -calculus in a logic programming language

Amy Felty, Dale Miller

UNITÉ DE RECHERCHE
INRIA-SOPHIA ANTIPOLIS

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél.:(1) 39 63 55 11

Rapports de Recherche

N° 1259

Programme 1
Programmation, Calcul Symbolique
et Intelligence Artificielle

ENCODING A DEPENDENT-TYPE λ-CALCULUS IN A LOGIC PROGRAMMING LANGUAGE

Amy FELTY
Dale MILLER

Juin 1990

RR-1259

Encoding a Dependent-Type λ-Calculus in a Logic Programming Language

Amy Felty
INRIA Sophia-Antipolis

Dale Miller
University of Pennsylvania

Abstract Various forms of typed λ-calculi have been proposed as specification languages for representing wide varieties of object logics. The *logical framework*, LF, is an example of such a dependent-type λ-calculus. A small subset of intuitionistic logic with quantification over simply typed λ-calculus has also been proposed as a framework for specifying general logics. The logic of *hereditary Harrop* formulas with quantification at all non-predicate types, denoted here as hh^ω, is such a meta-logic that has been implemented in both the Isabelle theorem prover and the λProlog logic programming language. In this paper, we show how LF can be encoded into hh^ω in a direct and natural way by mapping the typing judgments in LF into propositions in the logic of hh^ω. This translation establishes a strong connection between these two languages: the order of quantification in an LF signature is exactly the order of a set of hh^ω clauses, and the proofs in one system correspond directly to proofs in the other system.

Une Traduction d'un λ-Calcul avec Types Dépendants dans un Langage de Programmation Logique

Résumé Divers λ-calculs typés ont été proposés comme langages de spécification pour représenter une variété de logiques objets. Le système LF, *logical framework*, est un exemple de λ-calcul avec des types dépendants. Un sous-ensemble de la logique intuitionniste avec quantification sur les types simples a été proposé aussi comme méta-langage pour la spécification de logiques générales. La logique de formules *héréditaires de Harrop* avec quantification sur tous les types non-prédicats, dénotée hh^ω, est une méta-logique implémentée dans le démonstrateur Isabelle et le langage de programmation logique λProlog. Dans cet article, nous démontrons que LF peut être traduit dans hh^ω d'une manière directe et naturelle, en associant les jugements de types de LF aux propositions de la logique de hh^ω. Cette traduction établit une connexion forte entre les deux langages: l'ordre de quantification d'une signature LF est exactement le même que l'ordre d'un ensemble de formules hh^ω, et les preuves d'un système correspondent directement aux preuves de l'autre.

Encoding a Dependent-Type λ-Calculus in a Logic Programming Language *

Amy Felty
INRIA Sophia-Antipolis
2004, Route des Lucioles
06565 Valbonne Cedex, France

Dale Miller
Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104–6389 USA

Abstract

Various forms of typed λ-calculi have been proposed as specification languages for representing wide varieties of object logics. The *logical framework*, LF, is an example of such a dependent-type λ-calculus. A small subset of intuitionistic logic with quantification over simply typed λ-calculus has also been proposed as a framework for specifying general logics. The logic of *hereditary Harrop* formulas with quantification at all non-predicate types, denoted here as hh^ω, is such a meta-logic that has been implemented in both the Isabelle theorem prover and the λProlog logic programming language. Both frameworks provide for specifications of logics in which details involved with free and bound variable occurrences, substitutions, eigenvariables, and the scope of assumptions within object logics are handled correctly and elegantly at the "meta" level. In this paper, we show how LF can be encoded into hh^ω in a direct and natural way by mapping the typing judgments in LF into propositions in the logic of hh^ω. This translation establishes a very strong connection between these two languages: the order of quantification in an LF signature is exactly the order of a set of hh^ω clauses, and the proofs in one system correspond directly to proofs in the other system. Relating these two languages makes it possible to provide implementations of proof checkers and theorem provers for logics specified in LF by using standard logic programming techniques which can be used to implement hh^ω.

1 Introduction

The design and construction of computer systems that can be used to specify and implement large collections of logics has been the goal of several different research projects. In this paper we shall focus on two approaches to designing such systems. One approach is based on the use of dependent-type λ-calculi as a meta-language while another approach is based on the use of a very simple intuitionistic logic as a meta-language. The Logical

*To appear in the proceedings of the "10th International Conference on Automated Deduction," July 1990.

Framework (LF) [HHP87] and the Calculus of Constructions (CC) [CH88] are two examples of dependent-type calculi that have been proposed as meta-logics. The Isabelle theorem prover [Pau89] and the λProlog logic programming language [NM88] provide implementations of a common subset of intuitionistic logic, called hh^ω here, that can be used to specify a wide range of logics. Both Isabelle and λProlog can turn specifications of logics into proof checkers and theorem provers by making use of the unification of simply typed λ-terms and goal-directed, tactic-style search.

In this paper, we shall show that these two meta-languages are essentially of the same expressive power. This is done by showing how to translate LF specifications and judgments into a collection of hh^ω formulas such that correct typing in LF corresponds to intuitionistic provability in hh^ω. Besides answering the theoretical question about the precise relationship between these meta-languages, this translation also describes how LF specifications of an object logic can be implemented using unification and goal-directed search since these techniques provide implementations of hh^ω.

In Section 2 we present the meta-logic hh^ω and in Section 3 we present LF. Section 4 presents a translation of LF into hh^ω and Section 5 contains a proof of its correctness. Section 6 provides examples of this translation and Section 7 concludes.

2 The Meta-Logic

Let S be a fixed, finite set of *primitive types* (also called *sorts*). We assume that the symbol o is always a member of S. Following Church [Chu40], o is the type for propositions. The set of *types* is the smallest set of expressions that contains the primitive types and is closed under the construction of function types, denoted by the binary, infix symbol $\rightarrow$. The Greek letters τ and σ are used as syntactic variables ranging over types. The type constructor $\rightarrow$ associates to the right. If τ_0 is a primitive type then the type $\tau_1 \rightarrow \cdots \rightarrow \tau_n \rightarrow \tau_0$ has $\tau_1, \ldots, \tau_n$ as *argument types* and τ_0 as *target type*. The *order* of a primitive type is 0 while the order or a non-primitive type is one greater than the maximum order of its argument types.

For each type τ, we assume that there are denumerably many constants and variables of that type. Constants and variables do not overlap and if two constants (or variables) have different types, they are different constants (or variables). A *signature* is a finite set Σ of constants and variables whose types are such that their argument types do not contain o. A constant with target type o is a *predicate constant*. We often enumerate signatures by listing their members as pairs, written $a{:}\tau$, where a is a constant of type τ. Although attaching a type in this way is redundant, it makes reading signatures easier.

Simply typed λ-terms are built in the usual way. The logical constants are given the following types: $\wedge$ (conjunction) and $\supset$ (implication) are both of type $o \rightarrow o \rightarrow o$; $\top$ (true) is of type o; and $\forall_\tau$ (universal quantification) is of type $(\tau \rightarrow o) \rightarrow o$, for all types τ not containing o. A formula is a term of type o. The logical constants $\wedge$ and $\supset$ are written in the familiar infix form. The expression $\forall_\tau(\lambda z\ t)$ is written simply as $\forall_\tau z\ t$.

If x and t are terms of the same type then $[t/x]$ denotes the operation of substituting t for all free occurrences of x, systematically changing bound variables in order to avoid variable capture. The expression $[t_1/x_1, \ldots, t_n/x_n]$ will denote the simultaneous substitution of the terms $t_1, \ldots, t_n$ for the variables $x_1, \ldots, x_n$, respectively.

We shall assume that the reader is familiar with the usual notions and properties of α, β, and η conversion for the simply typed λ-calculus. The relation of convertibility up to α and β is written as $=_\beta$, and if η is added, is written as $=_{\beta\eta}$. We say that a λ-term is in β-normal form if it contains no beta redexes, that is, subformulas of the form $(\lambda x\ t)s$. A λ-term is in $\beta\eta$-long form if it is of the form

$$\lambda x_1 \ldots \lambda x_n (h t_1 \ldots t_m) \qquad (n, m \geq 0)$$

where h, called the head of the term, is either a constant or a variable, where the expression $ht_1 \ldots t_m$ is of primitive type, and where each term $t_1, \ldots, t_m$ are also in $\beta\eta$-long form. All λ-terms $\beta\eta$-convert to a term in $\beta\eta$-long form, unique up to α-conversion. See [HS86] for a fuller discussion of these basic properties of the simply typed λ-calculus.

Let Σ be a signature. A term is a Σ-*term* if all of its free variables and nonlogical constants are members of Σ. Similarly, a formula is a Σ-*formula* if all of its free variables and nonlogical constants are members of Σ. A formula is either *atomic* or *non-atomic*. An atomic Σ-formula is of the form $(Pt_1 \ldots t_n)$, where $n \geq 0$, P is given type $\tau_1 \to \cdots \to \tau_n \to o$ by Σ, and $t_1, \ldots, t_n$ are terms of the types $\tau_1, \ldots, \tau_n$, respectively. The predicate constant P is the *head* of this atomic formula. Non-atomic formulas are of the form $\top$, $B_1 \wedge B_2$, $B_1 \supset B_2$, or $\forall_\tau x\ B$, where B, B_1, and B_2 are formulas and τ is a type not containing o.

The logic we have just presented is very closely related to two logic programming extensions that have been studied elsewhere [MNPS]. *First-order hereditary Harrop* formulas (*fohh*) have been studied as an extension to first-order Horn clauses as a basis for logic programming. Similarly *higher-order hereditary Harrop* formulas (*hohh*) are a generalization of *fohh* that permits some forms of predicate quantification. Because our meta-language is neither higher-order, since it lacks predicate quantification, nor first-order, since it contains quantification at all function types, we shall simply call it hh^ω. The set of hh^ω formulas in which quantification only up to order n is used will be labeled as hh^n.

Provability for hh^ω can be given in terms of sequent calculus proofs. A *sequent* is a triple $\Sigma\ ;\ \mathcal{P} \longrightarrow B$, where Σ is a signature, B is a Σ-formula, and $\mathcal{P}$ is a finite (possibly empty) sets of Σ-formulas. The set $\mathcal{P}$ is this sequent's *antecedent* and B is its *succedent*. The expression $B, \mathcal{P}$ denotes the set $\mathcal{P} \cup \{B\}$; this notation is used even if $B \in \mathcal{P}$. The inference rules for sequents are presented in Figure 1. The following provisos are also attached to the two inference rules for quantifier introduction: in $\forall$-R the constant c is not in Σ, and in $\forall$-L t is a Σ-term of type τ.

A *proof* of the sequent $\Sigma\ ;\ \mathcal{P} \longrightarrow B$ is a finite tree constructed using these inference rules such that the root is labeled with $\Sigma\ ;\ \mathcal{P} \longrightarrow B$ and the leaves are labeled with *initial sequents*, that is, sequents $\Sigma'\ ;\ \mathcal{P}' \longrightarrow B'$ such that either B' is $\top$ or $B' \in \mathcal{P}'$. The non-terminals in such a tree are instances of the inference figures in Figure 1. Since we do not have an inference figure for $\beta\eta$-conversion, we shall assume that in building a proof, two formulas are equal if they are $\beta\eta$-convertible. If the sequent $\Sigma\ ;\ \mathcal{P} \longrightarrow B$ has a sequent proof then we write $\Sigma; \mathcal{P} \vdash_I B$ and say that B is provable from Σ and $\mathcal{P}$. We shall need only one proof-theoretic result concerning the meta-logic hh^ω. To state it, we require the following definition.

Definition 2.1 Let Σ be a signature and let $\mathcal{P}$ be a finite set of Σ-formulas. The

$$\frac{\Sigma\,;\, B, C, \mathcal{P} \longrightarrow C}{\Sigma\,;\, B \wedge C, \mathcal{P} \longrightarrow C}\ \wedge\text{-L} \qquad \frac{\Sigma\,;\, \mathcal{P} \longrightarrow B \quad \Sigma\,;\, \mathcal{P} \longrightarrow C}{\Sigma\,;\, \mathcal{P} \longrightarrow B \wedge C}\ \wedge\text{-R}$$

$$\frac{\Sigma\,;\, \mathcal{P} \longrightarrow B \quad \Sigma\,;\, C, \mathcal{P} \longrightarrow A}{\Sigma\,;\, B \supset C, \mathcal{P} \longrightarrow A}\ \supset\text{-L} \qquad \frac{\Sigma\,;\, B, \mathcal{P} \longrightarrow C}{\Sigma\,;\, \mathcal{P} \longrightarrow B \supset C}\ \supset\text{-R}$$

$$\frac{\Sigma\,;\, [t/x]B, \mathcal{P} \longrightarrow C}{\Sigma\,;\, \forall_\tau x\, B, \mathcal{P} \longrightarrow C}\ \forall\text{-L} \qquad \frac{\Sigma \cup \{c{:}\tau\}\,;\, \mathcal{P} \longrightarrow [c/x]B}{\Sigma\,;\, \mathcal{P} \longrightarrow \forall_\tau x\, B}\ \forall\text{-R}$$

Figure 1: Left and right introduction rules for hh^ω

expression $|\mathcal{P}|_\Sigma$ denotes the smallest set of pairs $\langle \mathcal{G}, D\rangle$ of finite sets of Σ-formulas $\mathcal{G}$ and Σ-formula D, such that

- If $D \in \mathcal{P}$ then $\langle \emptyset, D\rangle \in |\mathcal{P}|_\Sigma$.
- If $\langle \mathcal{G}, D_1 \wedge D_2\rangle \in |\mathcal{P}|_\Sigma$ then $\langle \mathcal{G}, D_1\rangle \in |\mathcal{P}|_\Sigma$ and $\langle \mathcal{G}, D_2\rangle \in |\mathcal{P}|_\Sigma$.
- If $\langle \mathcal{G}, \forall_\tau x D\rangle \in |\mathcal{P}|_\Sigma$ then $\langle \mathcal{G}, [t/x]D\rangle \in |\mathcal{P}|_\Sigma$ for all Σ-terms t of type τ.
- If $\langle \mathcal{G}, G \supset D\rangle \in |\mathcal{P}|_\Sigma$ then $\langle \mathcal{G} \cup \{G\}, D\rangle \in |\mathcal{P}|_\Sigma$.

Theorem 2.2 A non-deterministic search procedure for hh^ω can be organized using the following four search primitives.

AND: $B_1 \wedge B_2$ is provable from Σ and $\mathcal{P}$ if and only if both B_1 and B_2 are provable from Σ and $\mathcal{P}$.

GENERIC: $\forall_\tau x B$ is provable from Σ and $\mathcal{P}$ if and only if $[c/x]B$ is provable from $\Sigma \cup \{c : \tau\}$ and $\mathcal{P}$ for any constant $c : \tau$ not in Σ.

AUGMENT: $B_1 \supset B_2$ is provable from Σ and $\mathcal{P}$ if and only if B_2 is provable from Σ and $\mathcal{P} \cup \{B_1\}$.

BACKCHAIN: The atomic formula A is provable from Σ and $\mathcal{P}$ if and only if there is a pair $\langle \mathcal{G}, A\rangle \in |\mathcal{P}|_\Sigma$ so that for every $G \in \mathcal{G}$, G is provable from Σ and $\mathcal{P}$.

These formal results are closely related to the notion of *expanded normal form* for natural deduction proofs [Pra71] which was used by Paulson in [Pau89] to establish the correctness of a specification of first-order logic in hh^ω. This theorem will similarly play a central role in proving the correctness of our representation of LF in hh^ω.

3 The Logical Framework

There are three levels of terms in the LF type theory: objects (often called just terms), types and families of types, and kinds. We assume two given denumerable sets of variables, one for object-level variables and the other for type family-level variables. The

syntax of LF is given by the following classes of objects.

$$\begin{array}{rcl} K & := & \mathsf{Type} \mid \Pi x{:}A.K \\ A & := & x \mid \Pi x{:}A.B \mid \lambda x{:}A.B \mid AM \\ M & := & x \mid \lambda x{:}A.M \mid MN \\ \Gamma & := & \langle\rangle \mid \Gamma, x{:}K \mid \Gamma, x{:}A \end{array}$$

Here M and N range over expressions for objects, A and B over types and families of types, K over kinds, x over variables, and Γ over contexts. The empty context is denoted by $\langle\rangle$. We will use P and Q to range over arbitrary objects, types, type families, or kinds. We write $A \to P$ for $\Pi x{:}A.P$ when x does not occur in type or kind P. We will say that a type or type family of the form $xN_1 \ldots N_n$ where $n \geq 0$ and x is a type family-level variable is a *flat type*.

Terms that differ only in the names of variables bound by λ or Π are identified. If x is an object-level variable and N is an object then $[N/x]$ denotes the operation of substituting N for all free occurrences of x, systematically changing bound variables in order to avoid variable capture. The expression $[N_1/x_1, \ldots, N_n/x_n]$ will denote the simultaneous substitution of the terms $N_1, \ldots, N_n$ for distinct variables $x_1, \ldots, x_n$, respectively.

The notion of β-conversion at the level of objects, types, type families, and kinds can be defined in the obvious way using the usual rule for β-reduction at the level of both objects and type families: $(\lambda x{:}A.P)N \to_\beta [N/x]P$ where P is either an object or type/type family. The relation of convertibility up to β is written as $=_\beta$, just as it is at the meta-level. All well-typed LF terms are strongly normalizing [HHP87]. We write P^β to denote the normal form of term P. We present a version of the LF proof system that constructs only terms in canonical form, a notion which corresponds to $\beta\eta$-long forms in the simply typed λ-calculus. Several definitions from [HHP89] are required to establish this notion. We define the *arity* of a type or kind to be the number of Πs in the prefix of its normal form. The arity of a variable with respect to a context is the arity of its type in that context. The arity of a bound variable occurrence in a term is the arity of the type label attached to its binding occurrence. An occurrence of a variable x in a term is *fully applied* with respect to a context if it occurs in a subterm of the form $xM_1 \ldots M_n$, where n is the arity of x. A term P is *canonical* with respect to a context Γ if P is in β-normal form and every variable occurrence in P is fully applied with respect to Γ. We say that a context Γ is in canonical form if for every item $x : P$ in Γ, P is in canonical form with respect to Γ. Flat types of the form $xN_1 \ldots N_n$ such that x is fully applied will be called *base types*.

The following three kinds of *assertions* are derivable in the LF type theory.

$$\begin{array}{ll} \Gamma \vdash K \text{ kind} & (K \text{ is a kind in } \Gamma) \\ \Gamma \vdash A : K & (A \text{ has kind } K \text{ in } \Gamma) \\ \Gamma \vdash M : A & (M \text{ has type } A \text{ in } \Gamma) \end{array}$$

We write $\Gamma \vdash \alpha$ for an arbitrary assertion, where α is called an LF *judgment*. For the special form $\Gamma \vdash A : \mathsf{Type}$ of the second type of assertion, we also say A is a type in Γ. A context $x_1 : P_1 \ldots, x_n : P_n$ is said to be *valid* if $x_1, \ldots, x_n$ are distinct variables and for $i = 1, \ldots, n$, P_i is either a type or kind in context $x_1 : P_1, \ldots, x_{i-1} : P_{i-1}$. In

deriving one of the above assertions, we always assume that we start with a valid context Γ. We sometimes refer to the context in such an assertion as a *signature*.[1] Generally, a signature is a set of variables paired with types or kinds that specify an object logic and its inference rules.

The inference rules of LF are given in Figure 2. In (APP-OBJ) B must be a base type,

$$\Gamma \vdash \mathsf{Type}\ \mathrm{kind} \quad \text{(TYPE-KIND)}$$

$$\frac{\Gamma \vdash A : \mathsf{Type} \qquad \Gamma, x{:}A \vdash K\ \mathrm{kind}}{\Gamma \vdash \Pi x{:}A.K\ \mathrm{kind}}\ \text{(PI-KIND)}$$

$$\frac{\Gamma \vdash A : \mathsf{Type} \qquad \Gamma, x{:}A \vdash B : \mathsf{Type}}{\Gamma \vdash \Pi x{:}A.B : \mathsf{Type}}\ \text{(PI-FAM)}$$

$$\frac{\Gamma \vdash A : \mathsf{Type} \qquad \Gamma, x{:}A \vdash B : K}{\Gamma \vdash \lambda x{:}A.B : \Pi x{:}A.K}\ \text{(ABS-FAM)}$$

$$\frac{\Gamma \vdash A : \mathsf{Type} \qquad \Gamma, x{:}A \vdash M : B}{\Gamma \vdash \lambda x{:}A.M : \Pi x{:}A.B}\ \text{(ABS-OBJ)}$$

$$\frac{\begin{array}{l} x : \Pi x_1{:}A_1 \ldots \Pi x_n{:}A_n.\mathsf{Type} \in \Gamma \\ \Gamma \vdash N_1 : A_1 \\ \Gamma \vdash N_2 : ([N_1/x_1]A_2)^\beta \\ \vdots \\ \Gamma \vdash N_n : ([N_1/x_1, \ldots, N_{n-1}/x_{n-1}]A_n)^\beta \end{array}}{\Gamma \vdash x N_1 \ldots N_n : \mathsf{Type}}\ \text{(APP-FAM)}$$

$$\frac{\begin{array}{l} x : \Pi x_1{:}A_1 \ldots \Pi x_n{:}A_n.B \in \Gamma \\ \Gamma \vdash N_1 : A_1 \\ \Gamma \vdash N_2 : ([N_1/x_1]A_2)^\beta \\ \vdots \\ \Gamma \vdash N_n : ([N_1/x_1, \ldots, N_{n-1}/x_{n-1}]A_n)^\beta \end{array}}{\Gamma \vdash x N_1 \ldots N_n : ([N_1/x_1, \ldots, N_n/x_n]B)^\beta}\ \text{(APP-OBJ)}$$

Figure 2: The Logical Framework

and in (APP-OBJ) and (APP-FAM) n is the arity of x. In (PI-KIND), (PI-FAM), (ABS-FAM), and (ABS-OBJ), we assume that the variable x does not occur in Γ, and in (APP-FAM) and (APP-OBJ) we assume that the variables $x_1, \ldots, x_n$ do not occur free in $N_1, \ldots, N_n$. Note that bound variables can always be renamed to meet these restrictions.

The main difference between this presentation and the usual presentation of the LF proof system are the (APP-FAM) and (APP-OBJ) rules. The rules in the form presented here are those needed to preserve the invariant that all objects, types, type families, kinds, and contexts in derivable judgments are in canonical form. To see why, first note

[1] Other presentations of LF such as [HHP87] separate the notions of context and signature. We unify them here for simplicity of presentation.

that no new β-redexes are introduced in the conclusion of these rules. The application introduced in the left term of the judgment in the conclusion is always a variable applied to zero or more terms, while the right term is always β-normal. Second, note that the signature item x of arity n is applied in the conclusion to n terms and thus this occurrence of x is fully applied. Hence, as long as $N_1, \ldots, N_n$ are canonical, so is $xN_1 \ldots N_n$. In the (APP-OBJ) rule, the fact that the type $([N_1/x_1, \ldots, N_n/x_n]B)^\beta$ is canonical follows from the fact that for any object, type, type family, or kind P and any object M, if P and M are canonical, then so is $([M/x]P)^\beta$. Based on these observations, derivable assertions can be characterized more formally as follows: if Γ is a valid context, then Γ is canonical, and if $\Gamma \vdash \alpha$ is derivable with respect to valid context Γ, then the terms in α are canonical with respect to Γ, and α has one of the following forms.

1. $\Pi x_1 : A_1 \ldots \Pi x_n : A_n.\mathsf{Type}$ kind where $n \geq 0$.
2. $(\lambda x_1 : A_1 \ldots \lambda x_n : A_n.\Pi z_1 : B_1 \ldots \Pi z_m : B_m.C) : (\Pi x_1 : A_1 \ldots \Pi x_n : A_n.\mathsf{Type})$ where $n, m \geq 0$ and C is a base type.
3. $(\lambda x_1 : A_1 \ldots \lambda x_n : A_n.N) : (\Pi x_1 : A_1 \ldots \Pi x_n : A_n.B)$ where $n \geq 0$, N is not an abstraction, and B is a base type.

Note that proving an assertion of the form given by (2) or (3), respectively, in valid context Γ, is equivalent to proving $\Gamma, x_1 : A_1, \ldots, x_n : A_n \vdash \Pi z_1 : B_1 \ldots \Pi z_m : B_m.C : \mathsf{Type}$ or $\Gamma, x_1 : A_1, \ldots, x_n : A_n \vdash N : B$, respectively, in valid context $\Gamma, x_1 : A_1, \ldots, x_n : A_n$. In the first version of the translation given in the next section, we will assume that assertions have the latter form, *i.e.*, that there are no leading abstractions in the term on the left in a judgment.

4 Translating LF Assertions to hh^ω Formulas

In this section we present the translation of LF assertions to formulas in hh^ω. This translation will require an encoding of LF terms as simply typed λ-terms. We begin by presenting this encoding. We then present the translation, which given an LF assertion, $\Gamma \vdash \alpha$ where Γ is a valid context, translates Γ to a set of hh^ω formulas and α to a formula to be proved from this set of formulas. We then illustrate how to extend the translation to obtain a formula whose proof (from no assumptions) verifies that Γ is a valid context before proving that α holds within the context Γ.

Since both LF and the meta-language have types and terms, to avoid confusion we will refer to types and terms of the meta-language as *meta-types* and *meta-terms*. We only define the encoding of LF terms as simply typed λ-terms for LF objects and flat types/type families since this is all that is required by the translation. We introduce two base types, tm and ty, at the meta-level for these two classes of LF terms. First, to encode object-level variables, we define the function Φ that maps LF types and kinds to meta-types containing only occurrences of tm and ty.

$$\begin{aligned} \Phi(\Pi x : A.P) &:= \Phi(A) \to \Phi(P) \\ \Phi(\mathsf{Type}) &:= ty \\ \Phi(A) &:= tm \quad \text{when } A \text{ is a flat type} \end{aligned}$$

Using this function, an LF variable of kind or type P is mapped to a meta-variable of type $\Phi(P)$. These meta-types encode the "syntactic structure" of the corresponding LF dependent type or kind. Information about dependencies is lost in this mapping, but as we will see later, this information is retained in a different form in performing the general translation. We will assume a fixed mapping from LF variables to meta-variables of the corresponding type. For readability in our presentation, this mapping will be implicit. A variable x will represent both an LF variable with kind or type P and a meta-variable of the corresponding syntactic type $\Phi(P)$. It will always be clear from context which is meant. Note that for type or kind P and object N, $\Phi(P) = \Phi(([N/x]P)^{\beta})$.

We denote the encoding of term or flat type P as $\langle\!\langle P \rangle\!\rangle$. The full encoding is defined below.

$$\begin{array}{rcl} \langle\!\langle x \rangle\!\rangle & := & x \\ \langle\!\langle \lambda x{:}A.M \rangle\!\rangle & := & \lambda x{:}\Phi(A).\langle\!\langle M \rangle\!\rangle \\ \langle\!\langle MN \rangle\!\rangle & := & \langle\!\langle M \rangle\!\rangle \, \langle\!\langle N \rangle\!\rangle \\ \langle\!\langle AM \rangle\!\rangle & := & \langle\!\langle A \rangle\!\rangle \, \langle\!\langle M \rangle\!\rangle \end{array}$$

Note that the encoding maps abstraction in LF objects directly to abstraction at the meta-level, and that both application of objects to objects and application of type families to objects are mapped directly to application at the meta-level. The difference at the meta-level is that the former application will be a meta-term with target type tm while the latter application will be a meta-term with target type ty.

It is easy to see that for object or type family P having, respectively, type or kind Q, $\langle\!\langle P \rangle\!\rangle$ is a meta-term of meta-type $\Phi(Q)$. The following two properties also hold for this encoding.

Lemma 4.1 Let P be an LF object or flat type, and N an LF object. Then

$$[\langle\!\langle N \rangle\!\rangle / x]\langle\!\langle P \rangle\!\rangle = \langle\!\langle [N/x]P \rangle\!\rangle.$$

Lemma 4.2 Let P and Q be two LF objects or flat types. If $P =_{\beta} Q$, then $\langle\!\langle P \rangle\!\rangle =_{\beta} \langle\!\langle Q \rangle\!\rangle$.

We are now ready to define the translation. Two predicates will appear in the atomic hh^{ω} formulas resulting from the translation: *hastype* of type $tm \rightarrow ty \rightarrow o$ and *istype* of type $ty \rightarrow o$. We will name the signature containing these two predicates Σ_{LF}. We denote the translation of the context item or judgment α as $[\![\alpha]\!]$. The full translation is defined in Figure 3. It is a partial function since it is defined by cases and undefined when no case applies. It will in fact always be defined on contexts and judgments in provable LF assertions. In proving properties of the translation, we will only consider canonical judgments and context items. Note that in a canonical context item $x : P$, the variable x is not necessarily canonical since it may not be fully applied. Such judgments with non-canonical terms on the left are handled by the first and third rules in Figure 3. This translation maps occurrences of Π-abstraction in LF types and kinds directly to instances of universal quantification and implication in hh^{ω} formulas. In all of the clauses in the definition that contain a pattern with a Π-type or kind, the variable bound by Π is mapped to a variable at the meta-level bound by universal quantification. Then, in the resulting implication, the left hand side asserts the fact that the bound variable has a certain type, while the right hand side contains the translation of the body of the type or kind which may contain occurrences of this bound variable. The base cases occur when

$$
\begin{aligned}
[\![M : \Pi x{:}A.B]\!] &:= \forall_{\Phi(A)} x \left([\![x{:}A]\!] \supset [\![Mx : B]\!] \right) \\
[\![M : A]\!] &:= \mathit{hastype}\ \langle\!\langle M \rangle\!\rangle\ \langle\!\langle A \rangle\!\rangle \quad \text{where } A \text{ is a base type.} \\
[\![B : \Pi x{:}A.K]\!] &:= \forall_{\Phi(A)} x \left([\![x{:}A]\!] \supset [\![Bx : K]\!] \right) \\
[\![A : \mathsf{Type}]\!] &:= \mathit{istype}\ \langle\!\langle A \rangle\!\rangle \quad \text{where } A \text{ is a base type.} \\
[\![\Pi x{:}A.B : \mathsf{Type}]\!] &:= [\![A : \mathsf{Type}]\!] \wedge \forall_{\Phi(A)} x \left([\![x{:}A]\!] \supset [\![B : \mathsf{Type}]\!] \right) \\
[\![\mathsf{Type\ kind}]\!] &:= \top \\
[\![\Pi x{:}A.K\ \mathrm{kind}]\!] &:= [\![A : \mathsf{Type}]\!] \wedge \forall_{\Phi(A)} x \left([\![x{:}A]\!] \supset [\![K\ \mathrm{kind}]\!] \right)
\end{aligned}
$$

Figure 3: Translation of LF Judgments to hh^ω Formulas

there is no leading Π in the type or kind, resulting in atomic formulas for the *hastype* and *istype* predicates, or simply $\top$ in the case when the judgment is Type kind.

To illustrate this translation, we consider an example from an LF signature specifying natural deduction for first-order logic. The following declaration introduces the constant for universal quantification and gives it a type: $\forall^* : (i \to \mathit{form}) \to \mathit{form}$. (We write $\forall^*$ for universal quantification at the object level to distinguish it from universal quantification in hh^ω.) To make all bound variables explicit, we expand the above type to its unabbreviated form: $\Pi A{:}(\Pi y{:}i.\mathit{form}).\mathit{form}$. Note that by applying Φ to the above type, we get $(tm \to tm) \to tm$ as the type of $\forall^*$ at the meta-level. The translation of this signature item is as follows.

$$
\begin{aligned}
&[\![\forall^* : \Pi A{:}(\Pi y{:}i.\mathit{form}).\mathit{form}]\!] \equiv \\
&\quad \forall_{tm\to tm} A \left(\forall_{tm} y \left([\![y : i]\!] \supset [\![Ay : \mathit{form}]\!] \right) \supset [\![\forall^* A : \mathit{form}]\!] \right) \equiv \\
&\qquad \forall_{tm\to tm} A \left(\forall_{tm} y \left((\mathit{hastype}\ y\ i) \supset (\mathit{hastype}\ (Ay)\ \mathit{form}) \right) \supset (\mathit{hastype}\ (\forall^* A)\ \mathit{form}) \right)
\end{aligned}
$$

This formula provides the following description of the information contained in the above dependent type: for any A, if for arbitrary y of type i, Ay is a formula, then $\forall^* A$ is a formula.

We will show in the next section that if Γ is a valid canonical context and α a canonical judgment where the term on the left in α is not an abstraction, then $\Gamma \vdash \alpha$ is provable in LF iff $[\![\alpha]\!]$ is provable from the set of formulas $[\![\Gamma]\!]$. (Here $[\![\Gamma]\!]$ denotes the set of formulas obtained by translating separately each item in Γ.) We now illustrate how to extend the translation to obtain a formula whose proof verifies that Γ is a valid context before proving that α holds within the context Γ.

Proving that a context $x_1 : P_1, \ldots, x_n : P_n$ is valid in LF corresponds in hh^ω to proving, for $i = 1, \ldots, n$, either $[\![P_i : \mathsf{Type}]\!]$ or $[\![P_i\ \mathrm{kind}]\!]$ from $[\![x_1 : P_1, \ldots, x_{i-1} : P_{i-1}]\!]$. The translation in Figure 4, for an arbitrary assertion $\Gamma \vdash \alpha$, maps the pair $(\Gamma; \alpha)$ to a single formula containing subformulas whose proofs will in fact insure that each context item is valid with respect to the context items before it. We also remove the restriction that the term on the left in α cannot be an abstraction. Variables bound by abstraction

at the top level are treated as additional context items. The translation of such a pair is denoted $[\![\Gamma;\alpha]\!]^*$. The first two clauses of this translation map each context item to a

$$
\begin{aligned}
[\![x{:}A,\Gamma;\alpha]\!]^* &:= [\![A : \mathsf{Type}]\!] \wedge \forall_{\Phi(A)}x\ \left([\![x{:}A]\!] \supset [\![\Gamma;\alpha]\!]^*\right)\\
[\![x{:}K,\Gamma;\alpha]\!]^* &:= [\![K\ \mathrm{kind}]\!] \wedge \forall_{\Phi(K)}x\ \left([\![x{:}K]\!] \supset [\![\Gamma;\alpha]\!]^*\right)\\
[\![\langle\rangle;\lambda x{:}A.M : \Pi x{:}A.B]\!]^* &:= [\![A : \mathsf{Type}]\!] \wedge \forall_{\Phi(A)}x\ \left([\![x{:}A]\!] \supset [\![\langle\rangle;M : B]\!]^*\right)\\
[\![\langle\rangle;\lambda x{:}A.B : \Pi x{:}A.K]\!]^* &:= [\![A : \mathsf{Type}]\!] \wedge \forall_{\Phi(A)}x\ \left([\![x{:}A]\!] \supset [\![\langle\rangle;B : K]\!]^*\right)\\
[\![\langle\rangle;\alpha]\!]^* &:= [\![\alpha]\!] \quad \text{where the left term in } \alpha \text{ is not an abstraction.}
\end{aligned}
$$

Figure 4: Translation of LF Assertions

conjunctive formula where the first conjunct verifies that the type or kind is valid (using the previous translation), and the second conjunct is a universally quantified implication where the left hand side asserts the fact that the context item has the corresponding type (again using the previous translation), and the right side contains the translation of the pair consisting of the remaining context items and judgment. The third and fourth clauses handle the cases when the term on the left in a judgment is an abstraction. The last clause in the translation is for the base case: when the context is empty and no further abstractions remain at the head of the judgment. Then the previously defined translation is used. Thus, a proof of a formula obtained from translating an arbitrary assertion $\Gamma \vdash \alpha$ with respect to an initially empty set of assumptions verifies that each context item in Γ, and each variable bound by λ-abstraction in α is valid with respect to those items that appear before it, and then proves that the judgment holds within the entire context. The correctness of this translation will follow easily from the correctness of the previous translation.

5 Correctness of Translation

We consider the correctness of the translation with respect to a slightly modified LF. Our modified system replaces the (ABS-FAM) and (ABS-OBJ) rules with the following two rules.

$$\frac{\Gamma, x{:}A \vdash B : K}{\Gamma \vdash \lambda x{:}A.B : \Pi x{:}A.K}\ (\text{ABS-FAM}') \qquad \frac{\Gamma, x{:}A \vdash M : B}{\Gamma \vdash \lambda x{:}A.M : \Pi x{:}A.B}\ (\text{ABS-OBJ}')$$

These rules are the same as presented earlier except that the left premise is omitted. We call this system LF′. It can be shown that for valid context Γ, an LF assertion $\Gamma \vdash \alpha$ is provable in LF′ if and only if it is provable in LF, provided that there is no leading abstraction in the term on the left in α. This result relies on the fact that in a proof in the modified system, if an application of (ABS-FAM′) whose conclusion is $\Gamma \vdash \lambda x{:}A.B : \Pi x{:}A.K$ or an application of (ABS-OBJ′) whose conclusion is $\Gamma \vdash \lambda x{:}A.M : \Pi x{:}A.B$ occurs above an application of (APP-FAM) or (APP-OBJ), it is always the case that $\Gamma \vdash A : \mathsf{Type}$ is prov-

able, and thus the left premise is redundant. The proof of this fact relies on a transitivity result for LF′ similar to the one stated in [HHP87].

To prove the correctness of the translation, we prove a stronger statement from which the correctness of $[\![\ ]\!]$ and $[\![\ ]\!]^*$ will follow as corollaries. This stronger statement will talk about the provability of LF assertions of the form $\Gamma \vdash \alpha$ in LF′ even in the case when Γ is not a valid context. We also relax the requirement on α. The left term in α can be any object, type, type family, or kind, including one with a leading abstraction. To handle these cases in proving the correctness of $[\![\ ]\!]$, we must add the following two rules to the translation.

$$\begin{aligned}
[\![\lambda x{:}A.M : \Pi x{:}A.B]\!] &:= \forall_{\Phi(A)} x \left([\![x{:}A]\!] \supset [\![M : B]\!] \right) \\
[\![\lambda x{:}A.B : \Pi x{:}A.K]\!] &:= \forall_{\Phi(A)} x \left([\![x{:}A]\!] \supset [\![B : K]\!] \right)
\end{aligned}$$

We must then also add the restriction that the first and third rules in Figure 3 are only applicable when M and B, respectively, are not abstractions.

One final lemma is needed to prove the correctness of the translation. This lemma applies to the translation extended with the above two rules. In Section 4, we stated that substitution commutes with the encoding operation (Lemma 4.1). We extend this result to the translation operation on judgments which translate to provable hh^ω formulas. In particular, the lemma below states that substitution and β-normalization commute with the translation operation on provable hh^ω sequents. We will write $\Phi(\Gamma)$ to denote the set of meta-variables paired with their types obtained by mapping, for each signature item $x{:}P$ in Γ, the variable x to the corresponding meta-variable and P to $\Phi(P)$.

Lemma 5.1 Let $\Gamma, x_1 : A_1, \ldots, x_n : A_n, x : A$ $(n \geq 0)$ be a canonical context (whose variables are distinct). Let $N_1, \ldots, N_n, N$ be canonical objects with respect to Γ, and let Σ be the signature $\Sigma_{LF} \cup \Phi(\Gamma)$. Then $\Sigma; [\![\Gamma]\!] \vdash_I [\![N : ([N_1/x_1, \ldots, N_n/x_n]A)^\beta]\!]$ if and only if

$$\Sigma; [\![\Gamma]\!] \vdash_I [\langle\!\langle N_1 \rangle\!\rangle / x_1, \ldots, \langle\!\langle N_n \rangle\!\rangle / x_n, \langle\!\langle N \rangle\!\rangle / x][\![x : A]\!].$$

Theorem 5.2 (Correctness of Translation) Let Γ be an arbitrary context (such that the variables in Γ are distinct), and let α be an arbitrary canonical judgment with respect to Γ. Let Σ be $\Sigma_{LF} \cup \Phi(\Gamma)$. Then $\Gamma \vdash \alpha$ is provable in LF′ if and only if $\Sigma; [\![\Gamma]\!] \vdash_I [\![\alpha]\!]$ holds.

Proof Sketch: The proof of this theorem is constructive, *i.e.*, it provides a method for constructing an hh^ω proof from an LF′ proof, and vice versa. The forward direction is proved by induction on the height of an LF′ proof of the assertion $\Gamma \vdash \alpha$. For the PI and ABS rules, we can apply the induction hypothesis directly to the premises to obtain provable sequents to which we apply $\supset$-R, $\forall$-R, and $\wedge$-R for the PI rules, and just $\supset$-R and $\forall$-R for the ABS rules to obtain the desired result.

For the APP rules, we know the context item in the application of these rules corresponds to a formula in $[\![\Gamma]\!]$. To this formula, we can apply $\forall$-L followed by $\supset$-L n times in a backward direction. Each of the left premises of $\supset$-L can be shown to be provable since they are the result of applying the induction hypothesis followed by Lemma 5.1 to

each of the latter n premises of the APP rule. Using Lemmas 4.1 and 4.2, the formula introduced on the left in the right premise of the topmost application of $\supset$-L can be shown to be β-convertible to the formula in the succedent (the translation of the judgment in the conclusion of the APP rule). Thus this premise is an axiom.

The proof of the backward direction is by induction on the structure of the term on the left in α, and is similar to the proof of the forward direction. The regularity of the proofs in hh^ω described in Theorem 2.2 is required here. For example, the proof of the case when the term on the left is an application uses the backchain search operation. ■

Corollary 5.3 (Correctness of $[\![\ ]\!]$) Let Γ be a valid context and α a canonical judgment such that the term on the left is not an abstraction. Let Σ be $\Sigma_{LF} \cup \Phi(\Gamma)$. Then $\Gamma \vdash \alpha$ is provable in LF′ if and only if $\Sigma; [\![\Gamma]\!] \vdash_I [\![\alpha]\!]$ holds.

Corollary 5.4 (Correctness of $[\![\ ]\!]^*$) Let Γ be a canonical context (such that the variables in Γ are distinct), and α a canonical judgment. Then Γ is a valid context and $\Gamma \vdash \alpha$ is provable in LF′ if and only if $\Sigma_{LF}; \emptyset \vdash_I [\![\Gamma; \alpha]\!]^*$ holds.

6 Examples

In this section, we provide some further examples to illustrate the correspondence between LF signature items and judgments and the hh^ω formulas that they map to. Note that in general, formulas obtained by translating signature items have the form on the left below, but can be rewritten to have the form on the right:

$$\forall_{\tau_1} X_1\ (G_1 \supset \ldots \forall_{\tau_n} X_n\ (G_n \supset D) \ldots) \qquad \forall_{\tau_1} X_1 \ldots \forall_{\tau_n} X_n\ (G_1 \wedge \cdots \wedge G_n \supset D)$$

where $n \geq 0$, $\tau_1, \ldots, \tau_n$ are types, $X_1, \ldots, X_n$ are variables, $G_1, \ldots, G_n, D$ are hh^ω formulas. (Here we assume that for $i = 1, \ldots, n$, $X_{i+1}, \ldots, X_n$ do not appear free in G_i). For readability, we will write hh^ω formulas in the examples in this section simply as $G_1 \wedge \cdots \wedge G_n \supset D$ (or just D when $n = 0$), and assume implicit universal quantification over all free variables written as capital letters. Type subscripts for these universal quantifiers can always be inferred from context.

We begin by demonstrating the translation of signature items specifying natural deduction inference rules for the $\wedge^*$, $\forall^*$, and $\supset^*$ object-level connectives. The fragment of an LF signature specifying natural deduction for the first-order logic that we are concerned with is the following.

$$\begin{aligned}
i &: \mathsf{Type} \\
\mathit{form} &: \mathsf{Type} \\
\mathit{true} &: \mathit{form} \rightarrow \mathsf{Type} \\
\wedge^* &: \mathit{form} \rightarrow \mathit{form} \rightarrow \mathit{form} \\
\supset^* &: \mathit{form} \rightarrow \mathit{form} \rightarrow \mathit{form} \\
\forall^* &: (i \rightarrow \mathit{form}) \rightarrow \mathit{form} \\
\wedge^*\text{-I} &: \Pi A{:}\mathit{form}.\Pi B{:}\mathit{form}.\mathit{true}(A) \rightarrow \mathit{true}(B) \rightarrow \mathit{true}(A \wedge^* B) \\
\supset^*\text{-I} &: \Pi A{:}\mathit{form}.\Pi B{:}\mathit{form}.(\mathit{true}(A) \rightarrow \mathit{true}(B)) \rightarrow \mathit{true}(A \supset^* B) \\
\forall^*\text{-I} &: \Pi A{:}i \rightarrow \mathit{form}.(\Pi y{:}i.\mathit{true}(Ay)) \rightarrow \mathit{true}(\forall^* A)
\end{aligned}$$

The signature item *true* is a function that maps formulas to types. LF objects of type $true(A)$ represent proofs of formula A. First, consider the $\wedge^*$-introduction rule specified by $\wedge^*$-I and its type. Its translation is the following formula.

$$(hastype\ A\ form) \wedge (hastype\ B\ form) \wedge (hastype\ P\ (true\ A)) \wedge (hastype\ Q\ (true\ B)) \supset (hastype\ (\wedge^*\text{-I}\ A\ B\ P\ Q)\ (true\ A \wedge^* B))$$

This formula simply reads that if A and B have type *form*, P is a proof of A, and Q is a proof of B, then the term $(\wedge^*\text{-I}\ A\ B\ P\ Q)$ is a proof of the conjunction $A \wedge^* B$. The correspondence between this formula and the LF signature item is straightforward. We next consider a slightly more complex example; the translation of the $\forall^*$-I rule results in the following formula.

$$\forall y((hastype\ y\ i) \supset (hastype\ Ay\ form)) \wedge \forall y((hastype\ y\ i) \supset (hastype\ Py\ (true\ Ay))) \supset (hastype\ (\forall^*\text{-I}\ A\ P)\ (true\ \forall^* A))$$

This clause provides the following description of the information contained in the dependent type: if for arbitrary y of type i, Ay is a formula and Py is a proof of Ay, then the term $(\forall^*\text{-I}\ A\ P)$ is a proof of $\forall^* A$. Note that A and P at the meta-level are both functions having syntactic type $tm \rightarrow tm$. Here, A maps first-order terms to formulas just as it does at the object level, while P maps first-order terms to proofs. As a final inference rule example, consider the declaration for $\supset^*$-I, which translates to the following formula.

$$(hastype\ A\ form) \wedge (hastype\ B\ form) \wedge \forall q((hastype\ q\ (true\ A)) \supset (hastype\ Pq\ (true\ B))) \supset (hastype\ (\supset^*\text{-I}\ A\ B\ P)\ (true\ A \supset^* B))$$

This formula reads: if A and B are formulas and P is a function which maps an arbitrary proof q of A to the proof Pq of B, then the term $(\supset^*\text{-I}\ A\ B\ P)$ is a proof of $A \supset^* B$. Note that P in this formula is a function which maps proofs to proofs.

An example of a canonical judgment that is provable in the LF signature for natural deduction is

$$\lambda A{:}form.\supset^*\text{-I}(A)(A)(\lambda x{:}true(A).x)\ :\ \Pi A{:}form.true(A \supset^* A).$$

Using the extended translation of Figure 4, we obtain the following formula:

$$(istype\ form) \wedge \forall A((hastype\ A\ form) \supset (hastype\ (\supset^*\text{-I}\ A\ A\ \lambda x.x)\ (true\ A \supset^* A)))$$

which is provable from the set of formulas obtained by translating the LF signature specifying natural deduction for first-order logic.

An LF signature specifying the reductions needed for proof normalization in natural deduction is given in [Pfe89]. As a final example, we illustrate the translation of the reduction rule for the case when an application of the introduction rule for $\supset^*$ is followed by the elimination rule for the same connective. The following signature items define the $\supset^*$-E rule, the *reduce* constant used to relate two proofs of the same formula, and the reduction rule for $\supset^*$.

$$\begin{aligned} \supset^*\text{-E} &: \Pi A{:}form.\Pi B{:}form.true(A) \rightarrow true(A \supset^* B) \rightarrow true(B) \\ reduce &: \Pi A{:}form.true(A) \rightarrow true(A) \rightarrow \mathsf{Type} \\ \supset^*\text{-}red &: \Pi A{:}form.\Pi B{:}form.\Pi P{:}(true(A) \rightarrow true(B)).\Pi Q{:}true(A). \\ &\qquad reduce(B)(\supset^*\text{-E}\ A\ B\ (\supset^*\text{-I}\ A\ B\ P)\ Q)(PQ) \end{aligned}$$

The signature item for $\supset^*$-*red* translates to the following formula.

$$(\mathit{hastype}\ A\ \mathit{form}) \wedge (\mathit{hastype}\ B\ \mathit{form}) \wedge$$
$$\forall q\ ((\mathit{hastype}\ q\ (\mathit{true}\ A)) \supset (\mathit{hastype}\ (Pq)\ (\mathit{true}\ B))) \wedge (\mathit{hastype}\ Q\ (\mathit{true}\ A)) \supset$$
$$(\mathit{hastype}\ (\supset^*\text{-}\mathit{red}\ A\ B\ P\ Q)\ (\mathit{reduce}\ B\ (\supset^*\text{-E}\ A\ B\ (\supset^*\text{-I}\ A\ B\ P)\ Q)\ (PQ)))$$

This formula reads: if A and B are formulas and P is a function which maps an arbitrary proof q of A to the proof Pq of B, and Q is a proof of A, then $(\supset^*\text{-}\mathit{red}\ A\ B\ P\ Q)$ is a meta-proof of the fact that the natural deduction proof $(\supset^*\text{-E}\ A\ B\ (\supset^*\text{-I}\ A\ B\ P)\ Q)$ of B reduces to the proof PQ.

7 Conclusion

We have not yet considered the possibility of translating hh^ω formulas into LF. This translation is particularly simple. Let Σ be a signature for hh^ω and let $\mathcal{P}$ be a set of Σ-formulas. For each primitive type τ other than o in S, the corresponding LF judgment is τ : Type. For each non-predicate constant $c : \tau \in \Sigma$, the corresponding LF judgment is $c : \tau$. For each predicate constant $p : \tau_1 \to \cdots \to \tau_n \to o \in \Sigma$, the corresponding LF judgment is $p : \tau_1 \to \cdots \to \tau_n \to$ Type. Finally, let $D \in \mathcal{P}$ and let k be a new constant not used in the translation to this point. Then the corresponding LF judgment is $k : D'$ where D' is essentially D where $B_1 \supset B_2$ is written as $\Pi x{:}B_1.B_2$ and $\forall_\tau x\ B$ is written as $\Pi x{:}\tau.B$.

In the first author's dissertation [Fel89] an encoding of LF into just hh^2 was presented. Order 2 is all that is necessary if object-level applications are represented by meta-level constants. The proofs of the correctness of that encoding are very similar to those presented here.

Notice that the translation presented here works via recursion over the structure of types. Thus, this kind of translation will not work for the polymorphic λ-calculus or the Calculus of Constructions since they both contain quantification over types. Other techniques can be used, however, to encode provability of such λ-calculi into hh^ω. These involve coding the provability relation of those calculi directly into the meta-language [FM89].

Acknowledgements The authors would like to thank Robert Harper and Frank Pfenning for valuable discussions on the subject of this paper. We are also grateful to the reviewers of an earlier draft of this paper for their comments and corrections. Both authors have been supported in part by grants ONR N00014-88-K-0633, NSF CCR-87-05596, and DARPA N00014-85-K-0018. The first author is currently supported in part by ESPRIT Basic Research Action 3245.

References

[CH88] Thierry Coquand and Gérard Huet. The calculus of constructions. *Information and Computation*, 76(2/3):95–120, February/March 1988.

[Chu40] Alonzo Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.

[Fel89] Amy Felty. *Specifying and Implementing Theorem Provers in a Higher-Order Logic Programming Language.* PhD thesis, University of Pennsylvania, August 1989.

[FM89] Amy Felty and Dale Miller. A meta language for type checking and inference: an extended abstract. 1989. Presented at the 1989 Workshop on Programming Logic, Bålstad, Sweden.

[HHP87] Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. In *Second Annual Symposium on Logic in Computer Science*, pages 194–204, Ithaca, NY, June 1987.

[HHP89] Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. 1989. Technical Report CMU-CS-89-173, to appear.

[HS86] J. Roger Hindley and Jonathan P. Seldin. *Introduction to Combinatory Logic and Lambda Calculus.* Cambridge University Press, 1986.

[MNPS] Dale Miller, Gopalan Nadathur, Frank Pfenning, and Andre Scedrov. Uniform proofs as a foundation for logic programming. To appear in the Annals of Pure and Applied Logic.

[NM88] Gopalan Nadathur and Dale Miller. An overview of λProlog. In K. Bowen and R. Kowalski, editors, *Fifth International Conference and Symposium on Logic Programming*, MIT Press, 1988.

[Pau89] Lawrence C. Paulson. The foundation of a generic theorem prover. *Journal of Automated Reasoning*, 5(3):363–397, September 1989.

[Pfe89] Frank Pfenning. Elf: a language for logic definition and verified metaprogramming. In *Fourth Annual Symposium on Logic in Computer Science*, pages 313–321, Monterey, CA, June 1989.

[Pra71] Dag Prawitz. Ideas and results in proof theory. In J.E. Fenstad, editor, *Proceedings of the Second Scandinavian Logic Symposium*, pages 235–307, North-Holland, Amsterdam, 1971.

Imprimé en France
par
. l'Institut National de Recherche en Informatique et en Automatique .

ISSN 0249-6399